Pitman Programming Pocket Guides

Introduction to Programming	John Shelley
BASIC	Roger Hunt
COBOL	Ray Welland
FORTRAN	Philip Ridler
Pascal	David Watt
FORTRAN 77	Clive Page
Programming for the BBC Micro	Neil Cryer and Pat Cryer
Assembly Language for the 6502	Bob Bright

This series of pocket size reference guides to the major programming languages provides you with concise references and reliable descriptions of the main language features. You can use them as memory-joggers or reference tools.

An introduction to programming is available in this series for those who have had no programming experience. This provides you with the lead-in to the other programming language titles.

The Publishers would welcome suggestions for further improvements to this series. Please write to Alfred Waller at the address below.

PITMAN BOOKS LTD
128 Long Acre, London WC2E 9AN

Associated companies
Pitman Publishing Pty Ltd, Melbourne
Pitman Publishing New Zealand Ltd, Wellington
Copp Clark Pitman, Toronto

Consultant Editor: David Hatter

First edition, 1983

© Bob Bright, 1983

All rights reserved.

Printed in Great Britain

ISBN 0 273 01990 2

Index

How to use this Pocket Guide 1
Accumulator 3
Addressing
  absolute 6
  absolute indexed 8
  accumulator 10
  immediate 7–8
  implied 10
  indexed indirect 9
  indirect 9
  indirect indexed 9
  relative 10
  zero-page 7
    indexed 8
Assembler listing 6
Assembly language 5

BRK command 62

Control line
  CA1 52
  CA2 52
  CB1 53
  CB2 53
Control register
  auxiliary 53
CPU (6502) 1
  programming model 2

Data direction registers 51

Execution times 11–12

Flags 3–4
  break command 3
  carry 3
  decimal mode 3
  interrupt disable 3
  negative 4
  overflow 4
  testing 4
  zero 3

Index registers 51
Input/output 48
Input/output ports 51
Input registers 51
Instruction set 14–44
Internal registers 50
Interrupt
  enable register 59
  flag register 58–59
  non-maskable 60
  operation 59
  request 60
  return from 61
Interrupts 47, 58–62
  and reset 59

Language elements 4–6
Latches 53–57
Listing, assembler 6

Machine code 4–6

Operands 5–6
Operation Codes (see instruction set) 14–44
  Numerical sequence 42–44
Output registers 51

Paging system 12–13
Peripheral control register 52
Processor status register 3
Program counter 2
Programming model (see CPU) 2
Push and pull operations 45

Reset 59

Saving CPU status 61
Shift register 57
Source program 6
Stack manipulation 47
Stack pointer 2
  loading 45
Stack processes 44–48
Status register 3
Subroutines 45
  nested 46

Timer counters 54–56

Versatile Interface Adapter 48
  internal registers 50

How to use this Pocket Guide

Each feature of the 6502 assembly language has a section devoted to it.

Table I contains the complete instruction set and includes details of operation codes, various addressing modes, execution times and the required number of bytes of machine code. It also gives details of the processor status register operation.

Table II is a list of operation codes in numerical order together with the instruction mnemonic and the mode of addressing. There are sections on interrupts and stack processes together with a detailed study of the versatile interface adapter. Examples are given where necessary in the text and at the end of the guide.

6502 Central processing unit (CPU)

The 6502 is one of the family of the 6500 microprocessor devices. All the devices are software-compatible but not all are pin-compatible.

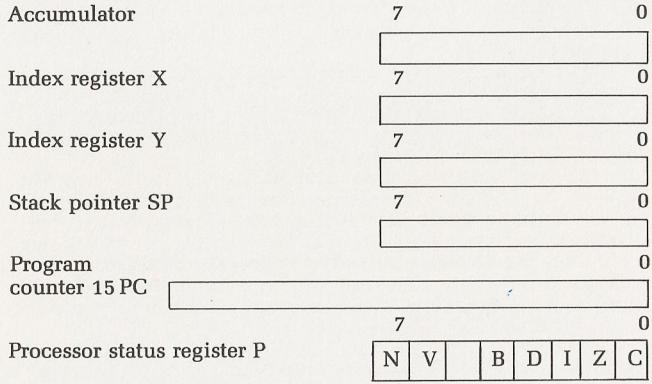
The 6502 is a 40-pin device and provides, among others, the following facilities:

56 instructions
13 addressing modes
16-bit address bus
8-bit bi-directional parallel data bus
1-MHz and 2-MHz system clock frequency
interrupt facilities—maskable and non-maskable
stack operation

The 6502 includes a number of registers, detailed in the section on the programming model of the CPU, an arithmetic logic unit, instruction decoder and register, and interrupt logic. The manufacturer's data sheet should be consulted for full details of the system architecture, electrical characteristics, pin connections, timing diagrams and clock operation.

Programming Model of the CPU

The programming model of the 6502 microprocessor is shown below.



N	negative flag
V	overflow flag
unused	
B	break command flag
D	decimal mode flag
I	interrupt disable bit
Z	zero flag
C	carry flag

Program counter PC

The program counter is a 16-bit register which contains the address of the next instruction or part instruction in the program sequence.

Stack pointer SP

The stack area is confined to page 01 of memory. The stack pointer holds the address of a memory location on this page and indicates the current position of the stack.

Accumulator

The accumulator is an 8-bit register used in certain arithmetic and logic operations; the results of such operations are placed in the accumulator.

Index registers X and Y

The index registers are 8-bit registers and are used in the various addressing modes involving indexed addressing. The registers can also be used in certain arithmetic operations and for transferring data between accumulator and the stack pointer.

Processor status register P

The processor status register is an 8-bit register containing 6 flags, an interrupt disable bit and an unused bit.

Bit 0 Carry flag (C)
The carry flag acts as a ninth bit in certain arithmetic and logic operations. Particular care must be used in the interpretation of this flag when using instructions involving subtraction. The flag should be cleared before starting any arithmetic operation using ADC and set before using SBC.

Bit 1 Zero flag (Z)
The zero flag is set if the result of arithmetic and logic operations is zero; if the result is non-zero the flag is cleared.

Bit 2 Interrupt disable bit (I)
This bit enables or disables $\overline{\text{IRQ}}$ interrupts:
if I = 1 $\overline{\text{IRQ}}$ interrupt is disabled;
if I = 0 $\overline{\text{IRQ}}$ interrupt is enabled.
It has no effect on $\overline{\text{NMI}}$ interrupts.

Bit 3 Decimal mode flag (D)
If the decimal mode flag is cleared the arithmetic unit within the CPU performs binary arithmetic; if the flag is set, binary coded decimal arithmetic is performed. The programmer should ensure that status of the flag is the one required.

Bit 4 Break command flag (B)
When a break instruction BRK is executed the CPU will set the break command flag (see BRK instruction p. 20).

Bit 5 Unused bit.

Bit 6 Overflow flag (V)
During signed binary arithmetic operations the overflow flag will be set if the magnitude of the result exceeds seven bits; otherwise it is cleared.

Bit 7 Negative flag (N)
The negative flag is set if the most significant bit of the result of an operation is also set; otherwise it is cleared.

The flags may be set, cleared or unaffected as a result of an instruction being executed; Table I shows the operation of the flags for each instruction and addressing mode.

Some of the flags are directly programmable:

CLC clear carry
CLD clear decimal mode
CLI clear interrupt disable bit
CLV clear overflow
SEC set carry
SED set decimal mode
SEI set interrupt disable bit

Testing the flags

Four of the flags namely N, V, Z and C can be tested and a conditional branch made depending on their status. Listed are the flags with the appropriate branch instructions:

N	BMI and BPL
V	BVS and BVC
Z	BEQ and BNE
C	BCS and BCC

Language elements

Machine code

The machine or object code for each instruction in the 6502 will consist of one, two or three bytes. The first byte always contains the operation code for the particular instruction together with the mode of addressing. For example, A2 is the code for LDX (load the index register X) using the immediate mode of addressing.

The second and third bytes, termed the operand, will if present, contain data, an address or a displacement from which an address can be formed.

Assembly language

The assembly language form for each instruction consists of an operator and an operand. The operator is represented by a mnemonic which is expressed by a three letter code which is meaningful to the user. If an assembler is used this code must be adhered to.

Example: LDA load accumulator
BEQ branch if equal

Operands

The format for the operand in the assembly language will, in addition to containing the data, etc, imply the mode of addressing. The formats are listed with their mode of addressing.

#Operand	immediate
Operand	zero page, absolute or relative
Operand, X	zero page and absolute indexed X
Operand, Y	zero page and absolute indexed Y
(Operand, X)	indexed indirect
(Operand), Y	indirect indexed
(Operand)	indirect
A	accumulator

If there is no Operand then the mode of addressing is implied.

The term operand in the previous list can be expressed in a number of ways:

1 As a binary, decimal, octal or hexadecimal number; it is usual to distinguish between the various number systems but the means will vary depending on the assembler used. Typical examples are:

hexadecimal	$80 or 80H
octal	@35 or 35Q
binary	% 01110001 or 01110001B
decimal	79 or 79D

2 As a symbol which will have to be defined and the assembler will assign the required numerical value.

3 As a simple arithmetic expression usually containing the operators +, −, / and *.

4 As an ASCII code character which is prefixed by a single quote (').

Source program

A source program is divided into four fields which contain respectively the label, operator, operand and comment. A typical source program is listed:

```
        ORG $3000
        LDY #$FF ; delay routine start
LOOP2   LDX #$FF
LOOP1   DEX
        BNE LOOP1
        DEY
        BNE LOOP2 ; delay routine end
```

The first field is occupied by a label and is optional; it is mainly used in branch and jump instructions. It assists the programmer to read and write programs and the assembler will calculate the offsets and numerical addresses. There is generally a restriction placed on the number and type of characters used to define a label and this will depend on the assembler used. The last field is reserved for comments to assist in the good documentation of programs and is entirely optional.

Assembler listing

An assembler will convert the source to object or machine code; a typical output from an assembler for the previous source program is listed below:

Location	Machine code	Label	Assembly language	Comments
3000	A0 FF		LDY #$FF	; delay routine start
3002	A2 FF	LOOP2	LDX #$FF	
3004	CA	LOOP1	DEX	
3005	D0 FD		BNE LOOP1	
3007	88		DEY	
3008	D0 F8		BNE LOOP2	; delay routine end

Addressing modes

Absolute addressing

Absolute addressing requires 3 bytes of machine code. The first byte contains the operation code; the second and third bytes

contain the low-order byte and the high-order bytes respectively of an address.

Example: LDA $2055
This instruction loads the accumulator with the contents of memory location $2055. Assembled into machine code this becomes

AD 55 20

The first byte AD is the operation code and the next two form the address of the memory location. Note that the 16-bit address is stored with the least significant byte first followed by the most significant byte.

Further example: JMP $3020
This instruction transfer program control to the memory location $3020. Assembled into machine code this becomes

4C 20 30

All 64k of addressable memory can be accessed with absolute addressing.

Zero-page addressing

Zero-page addressing requires 2 bytes of machine code. The first byte contains the operation code; second byte contains an address on page $00 of memory. The data to be operated on is held at this address.

Example: LDA $55
This instruction loads the accumulator with the contents of memory location $0055 or location $55 on page $00 of memory. Assembled into machine code this becomes

A5 55

The first byte A5 is the operation code and the second byte is the address on page $00.

Immediate addressing

Immediate addressing requires 2 bytes of machine code. The first byte contains the operation code; the second byte contains the data to be operated on.

Example: LDA #$9A
This instruction loads the accumulator with the data $9A. The # symbol specifies the immediate mode of addressing. Assembled into machine code this becomes

A9 9A

The first byte is the operation code and the second byte the data. In this mode of addressing it is said that the data immediately follows the operation code.

Absolute indexed addressing

Absolute indexed addressing requires 3 bytes of machine code. The first byte contains the operation code, the mode of addressing and the specified index register; the second and third bytes contain the low-order and high-order bytes of an address to which is added the contents of one of the index registers to form an effective address. The data to be operated on is held at this address.

Example: LDA $903A,X
If the index register X holds the value $30, then the accumulator is loaded with the contents of the memory location $906A ($903A+$30). Assembled into machine code this becomes

BD 3A 90

Further example: LDA $90A5,Y
If the index register Y holds the value $A9, then the accumulator is loaded with the contents of memory location $914E ($9035+$A9). Assembled into machine code this becomes

B9 A5 90

Note (a) that the 16-bit addresses are stored with the least significant byte first followed by the most significant byte, and (b) that there is no difficulty in crossing page boundaries but with some instructions the execution time will differ.

Zero-page indexed addressing

Zero-page indexed addressing requires 2 bytes of machine code. The first byte contains the operation code, the mode of addressing and the specified index register; the second byte contains an address in page $00 of memory to which is added the value held in the index register to form a zero-page address. The data to be operated is held at this address.

Example: LDA $20,X
If the index register X holds the value $3A, then the accumulator is loaded with the contents of the memory location $005A or location $5A on page $00 of memory. Assembled into machine code this becomes:

B5 20

Note that if the addition yields a value greater than $FF no carry is generated when forming the address. If, in the example, the index register held the value $F4 then the accumulator would be loaded

with the contents of the memory location $0014. This is commonly referred to as wrap-around.

Indirect addressing

Indirect addressing applies only to the Jump (JMP) instruction. It requires 3 bytes of machine code. The first byte contains the operation code, the second and third bytes contain the low-order and high-order bytes of a pointer address. The low-order and high-order bytes of the effective address are found at the pointer address.

Example: JMP ($2036)
If the contents of the locations $2036 and $2037 are $A9 and $54 respectively the program counter will contain $54A9 after the execution of the instruction.

2036	A9
2037	54

Indexed indirect addressing

Indexed indirect addressing requires 2 bytes of machine code. The first byte contains the operation code; the second byte is added to the value held in the index register to form a zero-page pointer address. The low-order and high-order bytes of the effective address are found at the pointer address. The data to be operated on is held at this effective address.

Example: LDA ($54,X)
If the index register X holds the value $38 and the contents of the memory locations $009C and $009D are $35 and $7B respectively then, as a result of this instruction, the contents of $7B35 will be loaded into the accumulator. Note that in forming the pointer address any carry out is ignored. Only the index register X can be used.

009C	35
009D	7B

Indirect indexed addressing

Indirect indexed addressing requires 2 bytes of machine code. The first byte contains the operation code; the second byte contains an address on page $00 of memory. The contents of this memory location are added to the contents of index register Y to form the

low-order byte of an effective address; any carry from this addition is added to the contents of the next memory location on page $00 of memory to form the high-order byte of the effective address. The data is held at this address.

Example: LDA ($A8),Y
At the pointer address and the pointer address+1, the low-order byte and the high-order byte of the effective address can be found. If the index register Y contains the value $30 and the memory locations $00A8 and $00A9 contains $37 and $15 respectively then, as a result of this instruction, the contents of $1567 ($1537+$30) are loaded into the accumulator.

00A8	37
00A9	15

Implied addressing

Implied addressing requires 1 byte of machine code.

Example: CLC
The carry flag is cleared as a result of this instruction. Assembled into machine code this becomes 18

Accumulator addressing

Accumulator addressing requires 1 byte of machine code; it applies to operations on the contents of the accumulator.

Example: ROL A
The contents of the accumulator and the carry flag are rotated 1 bit to the left. Assembled into machine code this becomes 2A.

Relative addressing

Relative addressing is used exclusively with the branch instructions and requires 2 bytes of machine code. When the branch instruction is executed the status of one of the processor flags is tested and causes a branch or otherwise, depending on the status of the flag. The first byte contains the instruction and the second byte contains a two's complement number which represents a displacement from the current position of the program counter.

Example: BEQ $06
In assembly language this becomes

Location	Machine code	Assembly language
PC	F0 06	BEQ $06
PC+2		

If the Z flag is set then the program counter is modified to (PC+2)+6 = PC+8; if the flag is clear then the program counter remains at PC+2.

Further example: BPL $FA
In assembly language this becomes

Location	Machine code	Assembly language
PC	10 FA	BPL $FA
PC+2		

If the N flag is clear the program counter changes to PC+2+(−6) = PC−4; if the flag is set the program counter remains at PC+2.

Since there is only 1 byte to define the displacement, the range of the branch is restricted to between -128_{10} and $+127_{10}$ from the current position of the program counter. If a conditional branch is required beyond this range, then a branch can be made to a location within the range and the JMP instruction used to jump to the required location.

Execution times

The execution time for each instruction is expressed in terms of the number of system clock cycles. Table I shows the number of clock cycles for each instruction; there are variations depending on the addressing mode.

The branch instruction in the table shows the execution time for the no branch condition. If a branch is made to the same page, then 1 cycle should be added to the no branch value and if a branch occurs to another page, 2 should be added.

Example:

Location	Label	Assembly language
2090		BPL LOOP
20A0	LOOP	

The instruction BPL tests the state of the N flag and causes a branch if N is clear.
If N = 1 execution time in cycles is 2;
if N = 0 execution time in cycles is 3 (branch to the same page)
However, if the address of the label is $2101 (another page), then if N=0, the time is 4 cycles.

Examination of Table I shows that the execution time for some of the instructions for certain addressing modes needs to be modified if a page boundary is crossed.

Example: single loop time delay with no page crossing

Location	Label	Assembly language	Comment
3000		LDX #$hh	; 2 cycles
3002	LOOP	DEX	; 2 cycles
3003		BNE LOOP	; 3 cycles for branch, 2 cycles otherwise

If $hh is equal to x_{10} then total number of clock cycles is

$$2 + 2x + 3(x-1) + 2 = 5x + 1$$

The LDX instruction is executed once, the DEX instruction x times and the branch occurs $x-1$ times; 2 cycles must be included for the no branch condition for BNE.

Further Example: double loop delay with page crossing for outer loop

Location	Machine code	Label	Assembly language		Comment
2FFC	A0 hh			LDY #$hh	; cycles 2
2FFE	A2 nn		LOOP2	LDX #$nn	; cycles 2
3000	CA		LOOP1	DEX	; cycles 2
3001	D0 FD			BNE LOOP 1	; cycles 3 or 2
3003	88			DEY	; cycles 2
3004	D0 F8			BNE LOOP2	; cycles 4 or 2

If $hh is equal to y_{10} and $nn equal to x_{10} then the total number of cycles is

$$y\ (5x+7)$$

The delay period is the number of cycles times the system clock period.

Paging system

The 6502 microprocessor can access up to 65536 or 64k memory locations using its 16-bit address bus; the addresses of the memory locations are normally expressed in terms of the hexadecimal number system. The total addressing rage is therefore from $0000 to $FFFF where $ indicates a hexadecimal number. Note that 1k is normally equivalent to 1024_{10}.

The memory of the 6502 is organized on a paging system; the most significant byte of the absolute address is the page number while the least significant indicates the location within that page.

For example, the address of memory location $3022 can be expressed as location $22 on page $30.

There are 256 pages with 256 bytes of memory per page.

There is very little restriction placed on the microcomputer designer in interfacing memory and input/output devices to the address bus. However it should be noted that pages $00, $01 and $FF are used for specific purposes.

page $00: the various index addressing modes use this page for storing data and addresses. Since the 6502 does not have a 16-bit index register, the power of the instruction set should not be diminished by using this page solely for the main program.

page $01: the stack area is restricted to this page. If the user's program involves subroutines and interrupts, it is advisable not to place the main-line program on this page for fear of the stack corrupting the program.

page $FF: the last six locations on this page are reserved for the addresses of the interrupt and reset service routines. In many microcomputer systems this page will probably contain the monitor program.

Key to Tables I and II

Accumulator	A
Index register X	X
Index register Y	Y
Memory	M
Processor status register	P
Stack pointer	S
Exclusive or	$\oplus$
Logical or	or
Logical and	.
Add	+
Subtract	−
Transfer to	$\rightarrow$
Transfer from	$\leftarrow$
Push on to stack	$\downarrow$
Pull from stack	$\uparrow$
Program counter	PC
Program counter high-order byte	PC$_{high}$
Program counter low-order byte	PC$_{low}$
Operand	Operand
Immediate	#

Processor status register

N	V		B	D	I	Z	C

Note: Heavy boxes round individual flags indicate possible changes.

Table I Instruction set

ADC: Add memory to accumulator with carry

Operation: A+M+C→ A,C

Description: The carry flag bit is added to the contents of the accumulator and memory; the result is placed in the accumulator.

Process status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
V: set if the addition results in a two s complement overflow; otherwise cleared.
Z: set if the result is zero; otherwise cleared.
C: set if there is a carry out from the most significant bit of the result; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	ADC #Operand	2	2	69
Zero Page	ADC Operand	3	2	65
Zero Page,X	ADC Operand,X	4	2	75
Absolute	ADC Operand	4	3	6D
Absolute,X	ADC Operand,X	4*	3	7D
Absolute,Y	ADC Operand,Y	4*	3	79
(Indirect,X)	ADC (Operand,X)	6	2	61
(Indirect),Y	ADC (Operand),Y	5*	2	71

*Add 1 if page boundary is crossed.

AND "And" memory with the accumulator

Operation: A.M →A

Description: a logical AND operation is performed between the corresponding bits of the memory and accumulator; the result is placed in the accumulator and the contents of the memory remains unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	AND #Operand	2	2	29
Zero Page	AND Operand	3	2	25
Zero Page,X	AND Operand,X	4	2	35
Absolute	AND Operand	4	3	2D
Absolute,X	AND Operand,X	4*	3	3D
Absolute,Y	AND Operand,Y	4*	3	39
(Indirect,X)	AND (Operand,X)	6	2	21
(Indirect),Y	AND (Operand),Y	5*	2	31

*Add 1 if the page boundary is crossed.

ASL Shift left One bit

Operation: C ← 7 6 5 4 3 2 1 0 ← 0

Description: the contents of memory or the accumulator are shifted left one bit; bit 0 is loaded with a logic 0 and bit 7 is shifted into the carry flag.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the memory or accumulator is set; otherwise cleared.
Z: set if the result of the memory or accumulator is zero; otherwise cleared.
C: set if as a result of the operation the carry flag is set; otherwise cleared.

Addressing Modes	Assembly Language	No. Cycles	No. Bytes	OP Code
Accumulator	ASL A	2	1	0A
Zero Page	ASL Operand	5	2	06
Zero Page,X	ASL Operand,X	6	2	16
Absolute	ASL Operand	6	3	0E
Absolute,X	ASL Operand,X	7	3	1E

BCC Branch if carry is clear

Description: the C flag is tested and a branch occurs if the flag is clear; otherwise the next instruction is executed.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Relative	BCC Operand	2*	2	90

* Add 1 if the branch occurs to the same page; add 2 if the branch occurs to a different page.

BCS Branch if carry is set

Description: the C flag is tested and a branch occurs if the flag is set; otherwise the next instruction is executed.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Relative	BCS Operand	2*	2	B0

*Add 1 if the branch occurs to the same page; add 2 if the branch occurs to a different page.

BEQ Branch if equal to zero

Description: The Z flag is tested and a branch occurs if the flag is set; otherwise the next instruction is executed.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Relative	BEQ Operand	2*	2	F0

*Add 1 if the branch occurs to the same page; add 2 if the branch occurs to a different page.

BIT Bit test

Operation: A.M, $M_7 \rightarrow N$ and $M_6 \rightarrow V$

Description: a logical AND operation is performed on the corresponding bits of the memory and accumulator; the contents of the memory and accumulator remain unchanged. Bits 7 and 6 of memory are transferred to the N and V flags respectively.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of memory is set; otherwise cleared.
Z: set if the result of the operation is zero; otherwise cleared.
V: set if bit 6 of the memory is set; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Zero Page	BIT Operand	3	2	24
Absolute	BIT Operand	4	3	2C

BMI Branch if minus

Description: the N flag is tested and a branch occurs if the flag is set; otherwise the next instruction is executed.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Relative	BMI Operand	2*	2	30

*Add 1 if the branch occurs to the same page; add 2 if the branch occurs to a different page.

BNE Branch if not equal to zero

Description: the Z flag is tested and a branch occurs if the flag is clear; otherwise the next instruction is executed.

Processor status register:
not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	Op Code
Relative	BNE Operand	2*	2	D0

*Add 1 if branch occurs to the same page; add 2 if the branch occurs to a different page.

BPL Branch if positive

Description: the N flag is tested and a branch occurs if the flag is cleared; otherwise the next instruction is executed.

Processor status register:
not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Relative	BPL Operand	2*	2	10

*Add 1 if branch occurs to the same page; add 2 if the branch occurs to a different page.

BRK Break command

Operation: PC + 2 ↓ , P ↓

Description: The break command flag is set. The program counter is then incremented by 2; the high-order byte, and the low-order byte of the program counter and the processor status register are all pushed onto the stack with the stack pointer being decremented each time. Program control is transferred to the break command interrupt service routine.

Processor status register:

N	V		B	D	I	Z	C

B: set
I: set

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	BRK	7	1	00

BVC Branch if overflow is cleared

Description: the V flag is tested and a branch occurs if the flag is clear; otherwise the next instruction is executed.

Processor status register:
not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Relative	BVC Operand	2*	2	50

*Add 1 if branch occurs to the same page; add 2 if branch occurs to a different page.

BVS Branch if overflow is set

Description: the V flag is tested and a branch occurs if the flag is set; otherwise the next instruction is executed.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Relative	BVS Operand	2*	2	70

*Add 1 if branch occurs to the same page; add 2 if branch occurs to a different page.

CLC Clear the carry flag

Operation: 0 → C

Description: the carry flag is cleared.

Processor status register:

N	V		B	D	I	Z	**C**

C: cleared

Addressing Mode	Assembly Language	No. Cycle	No. Bytes	OP Code
Implied	CLC	2	1	18

CLD Clear decimal mode

Operation: 0 → D

Description: the decimal mode flag is cleared.

Processor status register:

N	V		B	**D**	I	Z	C

D: cleared

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	CLD	2	1	D8

CLI Clear interrupt disable bit

Operation: 0 → I

Description: the interrupt disable bit is cleared.

Processor status register:

N	V		B	D	I	Z	C

I: cleared

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	CLI	2	1	58

CLV clear overflow flag

Operation: 0 → V

Description: the overflow flag is cleared.

Process status register:

N	V		B	D	I	Z	C

V: cleared

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	CLV	2	1	B8

CMP Compare memory and accumulator

Operation: A – M

Description: Compare by subtraction the contents of the memory with that of the accumulator; the contents of the accumulator and memory remain unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result of the comparison is set; otherwise cleared.
Z: set if the result of the comparison is zero; otherwise cleared.
C: cleared if the contents of the memory are greater than that of the accumulator; otherwise set.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	CMP #Operand	2	2	C9
Zero Page	CMP Operand	3	2	C5
Zero Page,X	CMP Operand,X	4	2	D5
Absolute	CMP Operand	4	3	CD
Absolute,X	CMP Operand,X	4*	3	DD
Absolute,Y	CMP Operand,Y	4*	3	D9
(Indirect,X)	CMP (Operand,X)	6	2	C1
(Indirect),Y	CMP (Operand),Y	5*	2	D1

*Add 1 if the page boundary is crossed.

CPX Compare memory and index register X

Operation: X–M

Description: Compare by subtraction the contents of the memory with that of the index register X; the contents of the memory and the index register X are unaffected.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the comparison is set; otherwise cleared.
Z: set if the result of the comparison is zero; otherwise cleared.
C: cleared if the contents of the memory are greater than the contents of the index register; otherwise set.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	CPX #Operand	2	2	E0
Zero Page	CPX Operand	3	2	E4
Absolute	CPX Operand	4	3	EC

CPY Compare memory and index register Y

Operation: Y–M

Description: Compare by subtraction the contents of the memory with that of the index register Y; the contents of the memory and the index register Y are unaffected.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the comparison is set; otherwise cleared.
Z: set if the result of the comparison is zero; otherwise cleared.
C: cleared if the contents of the memory are greater than the contents of the index register; otherwise set.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	CPX #Operand	2	2	C0
Zero Page	CPX Operand	3	2	C4
Absolute	CPX Operand	4	3	CC

DEC Decrement memory

Operation: M − 1 → M

Description: the contents of memory are decremented by 1.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the memory is set; otherwise cleared.
Z: set if the contents of the memory are zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Zero Page	DEC Operand	5	2	C6
Zero Page,X	DEC Operand,X	6	2	D6
Absolute	DEC Operand	6	3	CE
Absolute,X	DEC Operand,X	7	3	DE

DEX Decrement index register X

Operation: X−1→X

Description: the contents of the index register are decremented by 1.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the index register is set; otherwise cleared.
Z: set if the index register is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	DEX	2	1	CA

DEY Decrement index register Y

Operation: Y−1→Y

Description: the contents of the index register Y is decremented by 1.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the index register is set; otherwise cleared.
Z: set if the index register is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	DEY	2	1	88

EOR Exclusive or memory with accumulator

Operation: A⊕M → A

Description: an exclusive or operation is performed on the corresponding bits of the memory and accumulator; the result is placed in the accumulator, the contents of the memory are unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	EOR #Operand	2	2	49
Zero Page	EOR Operand	3	2	45
Zero Page,X	EOR Operand,X	4	2	55
Absolute	EOR Operand	4	3	4D
Absolute,X	EOR Operand,X	4*	3	5D
Absolute,Y	EOR Operand,Y	4*	3	59
(Indirect,X)	EOR (Operand,X)	6	2	41
(Indirect),Y	EOR (Operand),Y	5*	2	51

*Add 1 if the page boundary is crossed.

INC Increment memory

Operation: M+1→M

Description: the contents of the memory are incremented by 1.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Zero Page	INC Operand	5	2	E6
Zero Page,X	INC Operand,X	6	2	F6
Absolute	INC Operand	6	3	EE
Absolute,X	INC Operand,X	7	3	FE

INX Increment index register

Operation: X+1→X

Description: the contents of the index register are incremented by 1.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	INX	2	1	E8

INY Increment index register

Operation: Y+1→Y

Description: the contents of the index register Y are incremented by 1

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	INY	2	1	C8

JMP Jump

Operation: (PC+1) → PC_{low}, (PC+2)→ PC_{high}

Description: A jump occurs to the absolute address obtained from the operand.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Absolute	JMP Operand	3	3	4C
Indirect	JMP (Operand)	5	3	6C

JSR Jump to subroutine

Operation: PC + 2 ↓ , (PC+1)→ PC_{low} (PC+2)→PC_{high}

Description: The program counter is incremented by 2. The high-order byte is pushed onto the stack and the stack pointer is decremented by 1; then the low-order byte is pushed onto the stack and the stack pointer is decremented once more. A jump occurs to the absolute address contained in the operand.

Processor Status Register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Absolute	JSR Operand	6	3	20

LDA Load accumulator

Operation: M→A

Description: The contents of memory are loaded into the accumulator.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	LDA #Operand	2	2	A9
Zero Page	LDA Operand	3	2	A5
Zero page, X	LDA Operand,X	4	2	B5
Absolute	LDA Operand	4	3	AD
Absolute,X	LDA Operand,X	4*	3	BD
Absolute,Y	LDA Operand,Y	4*	3	B9
(Indirect,X)	LDA (Operand,X)	6	2	A1
(Indirect),Y	LDA (Operand),Y	5*	2	B1

*Add 1 if the page boundary is crossed.

LDX Load index register

Operation: M→X

Description: the contents of memory are loaded into the index register.

Process status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	LDX #Operand	2	2	A2
Zero Page	LDX Operand	3	2	A6
Zero Page,Y	LDX Operand,Y	4	2	B6
Absolute	LDX Operand	4	3	AE
Absolute,Y	LDX Operand,Y	4*	3	BE

*Add 1 when the page boundary is crossed.

LDY Load index register Y

Operation: M→Y

Description: the contents of memory are loaded into the index register.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
'Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	LDY #Operand	2	2	A0
Zero Page	LDY Operand	3	2	A4
Zero Page,X	LDY Operand,X	4	2	B4
Absolute	LDY Operand	4	3	AC
Absolute,X	LDY Operand, X	4*	3	BC

*Add 1 when the page boundary is crossed.

LSR Logic shift right

Operation: 0 → 7 6 5 4 3 2 1 0 → C

Description: the contents of the accumulator or memory are shifted right one bit; logic 0 is shifted into the most significant bit and bit 0 is shifted into the carry bit.

Processor status register:

N	V		B	D	I	Z	C

N: cleared.
Z: set if the result in the accumulator or memory is zero; otherwise cleared.
C: set if prior to the operation bit 0 was set; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Accumulator	LSR A	2	1	4A
Zero Page	LSR Operand	5	2	46
Zero page,X	LSR Operand,X	6	2	56
Absolute	LSR Operand	6	2	4E
Absolute,X	LSR Operand,X	7	3	5E

NOP No operation

Description: no operation for 2 cycles.

Processor status register: not affected.

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	NOP	2	1	EA

ORA 'OR' memory with accumulator

Operation: A or M → A

Description: a logical or operation is performed with the corresponding bits of the memory and the accumulator; the result is placed in the accumulator. The contents of the memory remain unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	ORA #Operand	2	2	09
Zero Page	ORA Operand	3	2	05
Zero Page,X	ORA Operand,X	4	2	15
Absolute	ORA Operand	4	3	0D
Absolute,X	ORA Operand,Y	4*	3	1D
Absolute,Y	ORA Operand,Y	4*	3	19
(Indirect,X)	ORA (Operand,X)	6	2	01
(Indirect),Y	ORA (Operand),Y	5*	2	11

*Add 1 if the page boundary is crossed.

PHA Push accumulator on stack

Operation: A ↓

Description: the contents of the accumulator are pushed onto the stack at the address contained in the stack pointer; the stack pointer is decremented by 1.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	PHA	3	1	48

PHP Push processor status register on stack

Operation: P↓

Description: the contents of the processor status register are pushed onto the stack at the address contained in the stack pointer; the stack pointer is decremented by 1.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	PHP	3	1	08

PLA Pull accumulator from stack

Operation: A ↑

Description: The stack pointer is incremented by 1; the accumulator is loaded with the contents of the memory location whose address is contained in the stack pointer.

Processor status register:

N	V		B	D	I	Z	C

N: Set if the most significant bit of the accumulator is set; otherwise cleared.
Z: Set if the contents of the accumulator are zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	PLA	4	1	68

PLP Pull processor status register from stack

Operation: P↑

Description: The stack pointer is incremented by 1; the processor status register is loaded with the contents of the memory location whose address is contained in the stack pointer.

Processor status register:

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	PLP	4	1	28

ROL Rotate left

Operation: C → 7 6 5 4 3 1 0 → C

Description: The contents of the memory or accumulator are rotated one bit left; the carry flag bit is rotated into bit 0 and the most significant bit is rotated into the carry flag.

Processor status register: not affected

N	V		B	D	I	Z	C

N: set if after rotation the most significant bit of the memory or accumulator is set; otherwise cleared.
Z: set if after rotation the contents of the memory or accumulator is zero; otherwise cleared.
C: set if the most significant bit of the memory or accumulator prior to rotation was set; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Accumulator	ROL A	2	1	2A
Zero Page	ROL Operand	5	2	26
Zero Page,X	ROL Operand,X	6	2	36
Absolute	ROL Operand	6	3	2E
Absolute,X	ROL Operand,X	7	3	3E

ROR Rotate right

Operation: C ← 7 6 5 4 3 2 1 0 ← C

Description: The contents of the memory or accumulator are rotated one bit right; the carry flag bit is rotated into the most significant bit of the memory or accumulator and bit 0 is rotated into the carry flag.

Processor status register:

N	V		B	D	I	Z	C

N: set if after rotation the most significant bit of the memory or accumulator is set; otherwise cleared.
Z: set if after rotation the contents of the memory or accumulator is zero; otherwise cleared.
C: set if the least significant bit of the memory or accumulator prior to rotation is set; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Accumulator	ROR A	2	1	6A
Zero Page	ROR Operand	5	2	66
Zero Page,X	ROR Operand,X	6	2	76
Absolute	ROR Operand	6	3	6E
Absolute,X	ROR Operand,X	7	3	7E

RTI Return from interrupt

Operation: P ↑ PC ↑

Description: The stack pointer is incremented by 1. The byte stored at the address contained in the stack pointer is loaded into the processor status register; the stack pointer is incremented by 1 and the byte stored at the address contained in the stack pointer is loaded into the low-order byte of the program counter. The stack is again incremented and the next byte loaded into the high-order byte of the program counter.

Processor status register:

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	RTI	6	1	40

RTS Return from subroutine

Operation: PC ↑ , PC + 1 → PC

Description: The stack pointer is incremented by 1. The byte stored at the address contained in the stack pointer is loaded into the low-order byte of the program counter; the stack pointer is again incremented by 1 and the byte stored at the address contained in the stack pointer is loaded into the high-order byte of the program counter. Then the program counter is incremented by 1.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	RTS	6	1	60

SBC Subtract memory from accumulator with carry

Operation: A − M − $\overline{C}$ → A,C

Description: The contents of memory plus the complement of the carry flag bit are subtracted from the contents of the accumulator; the result is placed in the accumulator.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the result is set; otherwise cleared.
Z: set if the result is zero; otherwise cleared.
C: cleared if the contents of memory plus the complement of the carry flag bit are greater than the contents of the accumulator; otherwise set.
V: set if the subtraction results in a two's complement overflow; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Immediate	SBC #Operand	2	2	E9
Zero Page	SBC Operand	3	2	E5
Zero Page,X	SBC Operand,X	4	2	F5
Absolute	SBC Operand	4	3	ED
Absolute,X	SBC Operand,X	4*	3	FD
Absolute,Y	SBC Operand,Y	4*	2	F9
(Indirect,X)	SBC (Operand,X)	6	2	E1
(Indirect),Y	SBC (Operand,Y	5*	2	F1

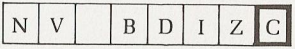
*Add 1 if page boundary is crossed.

SEC Set carry

Operation: 1→ C

Description: The carry flag is set.

Processor status register:



C: set

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	SEC	2	1	38

SED Set decimal mode

Operation: 1 → D

Description: The decimal mode flag is set

Processor status register:

D: set

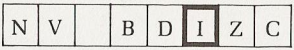
Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	SED	2	1	F8

SEI Set interrupt

Operation: 1 → I

Description: The interrupt disable bit is set.

Processor status register:



I: set

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	SEI	2	1	78

STA Store accumulator

Operation: A→M

Description: The contents of accumulator are stored in memory.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Zero Page	STA Operand	3	2	85
Zero Page,X	STA Operand	4	2	95
Absolute	STA Operand	4	3	8D
Absolute,X	STA Operand	5	3	9D
Absolute,Y	STA Operand	5	3	99
(Indirect,X)	STA (Operand,X)	6	2	81
(Indirect),Y	STA (Operand),Y	6	2	91

STX Store index X

Operation: X→M

Description: The contents of the index register X are stored in memory.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Zero Page	STX Operand	3	2	86
Zero Page,Y	STX Operand,Y	4	2	96
Absolute	STX Operand	4	3	8E

STY Store index Y

Operation: Y → M

Description: The contents of the index register Y are stored in memory.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Zero Page	STX Operand	3	2	84
Zero Page,X	STX Operand,X	4	2	94
Absolute	STX Operand	4	3	8C

TAX Transfer accumulator to index register X

Operation: A→X

Description: The contents of the accumulator are transferred to the index register X; the contents of the accumulator remain unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the accumulator is set; otherwise cleared.
Z: set if the contents of the accumulator are zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	TAX	2	2	AA

TAY Transfer accumulator to index register Y

Operation: A→Y

Description: The contents of the accumulator are transferred to the index register Y; the contents of the accumulator remain unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the accumulator is set; otherwise cleared.
Z: set if the contents of the accumulator are zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	TAY	2	1	A8

TYA Transfer index register Y to accumulator

Operation: Y→A

Description: The contents of the index register Y are transferred to the accumulator; the contents of the index register Y remain unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the accumulator is set; otherwise cleared.
Z: set if the contents of the accumulator are zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	TYA	2	1	98

TSX Transfer stack pointer to index X

Operation: S→X

Description: The contents of the stack pointer are transferred to the index register X; the contents of the stack pointer remain unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the stack pointer is set; otherwise cleared.
Z: set if the contents of the stack pointer are set; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	TSX	2	2	BA

TXA Transfer index X to accumulator

Operation: X→A

Description: The contents of the index register X are transferred to the accumulator; the contents of the index register remain unchanged.

Processor status register:

N	V		B	D	I	Z	C

N: set if the most significant bit of the index register is set; otherwise cleared.
Z: set if the contents of the index register are zero; otherwise cleared.

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	TXA	2	1	8A

TXS Transfer index X to stack pointer

Operation: X→S

Description: The contents of the index register X are transferred to the stack pointer; the contents of the index register remain unchanged.

Processor status register: not affected

N	V		B	D	I	Z	C

Addressing Mode	Assembly Language	No. Cycles	No. Bytes	OP Code
Implied	TXS	2	1	9A

Table II Operation codes listed in numerical sequence

Code	Inst.	Addressing Mode	Code	Inst.	Addressing Mode
00	BRK	Implied	28	PLP	Implied
01	ORA	(Indirect,X)	29	AND	Immediate
05	ORA	Zero page	2A	ROL	Accumulator
06	ASL	Zero page	2C	BIT	Absolute
08	PHP	Implied	2D	AND	Absolute
09	ORA	Immediate	2E	ROL	Absolute
0A	ASL	Accumulator	30	BMI	Relative
0D	ORA	Absolute	31	AND	(Indirect),Y
0E	ASL	Absolute	35	AND	Zero page,X
10	BPL	Relative	36	ROL	Zero page,X
11	ORA	(Indirect),Y	38	SEC	Implied
15	ORA	Zero page,X	39	AND	Absolute,Y
16	ASL	Zero page,X	3D	AND	Absolute,X
18	CLC	Implied	3E	ROL	Absolute,X
19	ORA	Absolute,Y	40	RTI	Implied
1D	ORA	Absolute,X	41	EOR	(Indirect,X)
1E	ASL	Absolute,X	45	EOR	Zero page
20	JSR	Absolute	46	LSR	Zero page
21	AND	(Indirect,X)	48	PHA	Implied
24	BIT	Zero page	49	EOR	Immediate
25	AND	Zero page	4A	LSR	Accumulator
26	ROL	Zero page	4C	JMP	Absolute

Code	Inst.	Addressing Mode	Code	Inst.	Addressing Mode
4D	EOR	Absolute	99	STA	Absolute,Y
4E	LSR	Absolute	9A	TXS	Implied
50	BVC	Relative	9D	STA	Absolute,X
51	EOR	(Indirect),Y	A0	LDY	Immediate
55	EOR	Zero page,X	A1	LDA	(Indirect,X)
56	LSR	Zero page,X	A2	LDX	Immediate
58	CLI	Implied	A4	LDY	Zero page
59	EOR	Absolute,Y	A5	LDA	Zero page
5D	EOR	Absolute,X	A6	LDX	Zero page
5E	LSR	Absolute,X	A8	TAY	Implied
60	RTS	Implied	A9	LDA	Immediate
61	ADC	(Indirect,X)	AA	TAX	Implied
65	ADC	Zero page	AC	LDY	Absolute
66	ROR	Zero page	AD	LDA	Absolute
68	PLA	Implied	AE	LDX	Absolute
69	ADC	Immediate	B0	BCS	Relative
6A	ROR	Accumulator	B1	LDA	(Indirect),Y
6C	JMP	Indirect	B4	LDY	Zero page,X
6D	ADC	Absolute	B5	LDA	Zero page,X
6E	ROR	Absolute	B6	LDX	Zero page,Y
70	BVS	Relative	B8	CLV	Implied
71	ADC	(Indirect),Y	B9	LDA	Absolute,Y
75	ADC	Zero page,X	BA	TSX	Implied
76	ROR	Zero page,X	BC	LDY	Absolute,X
78	SEI	Implied	BD	LDA	Absolute,X
79	ADC	Absolute,Y	BE	LDX	Absolute,Y
7D	ADC	Absolute,X	C0	CPY	Immediate
7E	ROR	Absolute,X	C1	CMP	(Indirect,X)
81	STA	(Indirect,X)	C4	CPY	Zero page
84	STY	Zero page	C5	CMP	Zero page
85	STA	Zero page	C6	DEC	Zero page
86	STX	Zero page	C8	INY	Implied
88	DEY	Implied	C9	CMP	Immediate
8A	TXA	Implied	CA	DEX	Implied
8C	STY	Absolute	CC	CPY	Absolute
8D	STA	Absolute	CD	CMP	Absolute
8E	STX	Absolute	CE	DEC	Absolute
90	BCC	Relative	D0	BNE	Relative
91	STA	(Indirect),Y	D1	CMP	(Indirect),Y
94	STY	Zero page,X	D5	CMP	Zero page,X
95	STA	Zero page,X	D6	DEC	Zero page,X
96	STX	Zero page,Y	D8	CLD	Implied
98	TYA	Implied	D9	CMP	Absolute,Y

Code	Inst.	Addressing Mode	Code	Inst.	Addressing Mode
DD	CMP	Absolute,X	ED	SBC	Absolute
DE	DEC	Absolute,X	EE	INC	Absolute
E0	CPX	Immediate	F0	BEQ	Relative
E1	SBC	(Indirect,X)	F1	SBC	(Indirect),Y
E4	CPX	Zero page	F5	SBC	Zero page,X
E5	SBC	Zero page	F6	INC	Zero page,X
E6	INC	Zero page	F8	SED	Implied
E8	INX	Implied	F9	SBC	Absolute,Y
E9	SBC	Immediate	FD	SBC	Absolute,X
EA	NOP	Implied	FE	INC	Absolute,X
EC	CPX	Absolute			

Stack processes

The stack is an area of random-access memory where data can be temporarily stored and retrieved by the CPU and the programmer. In the 6502 system the stack is restricted to page 01 of memory and since each page contains only 256 memory locations, an 8-bit register is sufficient to identify the current position of the stack. This register is called the *stack pointer*.

The stack operates in the following way. Data is pushed onto the stack one byte at a time at the address contained in the stack pointer; as each byte is pushed the stack pointer is decremented by 1 thus ensuring that the stack pointer points to the next 'empty' location in the stack. In retrieving data the stack pointer is incremented by 1 before the data is pulled from the stack at the address contained in the stack pointer.

Figure 1 shows a diagrammatic description of the operation. Assume initially that the stack pointer contains the value SP. Data is stored in the following sequence BYTE1, BYTE2 . . . BYTE5 with the stack pointer being decremented until finally containing the value SP-5. Data is retrieved in the reverse sequence BYTE5 . . . BYTE1 with the stack pointer finally containing SP.

This stack operation is commonly referred to as LIFO (last-in-first-out).

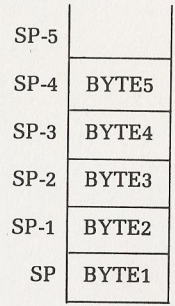


Fig. 1

Loading the stack pointer

During the initialization sequence of a program it is necessary to load the stack pointer with the required address. There is no instruction which loads the stack pointer directly

```
LDX #$FF   ; loads index register X with $FF
TXS        ; transfers the contents of the index register X to the
             stack pointer
```

The current position of the stack is $01FF. Note that the index register Y cannot be used for this operation.

Push and pull operations

There are four instructions which can be used to store and retrieve data from the stack. These are PHA, PHP, PLA and PLP.

PHA ; the contents of the accumulator are pushed onto the stack and the stack pointer is decremented by 1.
PLA ; the stack pointer is incremented by 1 and the data pulled from the stack is loaded into the accumulator.
PHP ; the contents of the processor status register are pushed onto the stack and the stack pointer is decremented by 1.
PLP; ; the stack pointer is incremented by 1 and the data pulled from the stack is loaded into the processor status register.

The contents of the index registers X and Y cannot be saved directly on the stack and the accumulator must be used.

```
TXA ; contents of index register X transfered to A
PHA ; and pushed on to the stack.
```

For retrieval

```
PLA ; contents of the stack pulled from the stack
TAX ; into A and transferred to X.
```

By using TYA and TAY the status of index register Y can also be saved and retrieved.

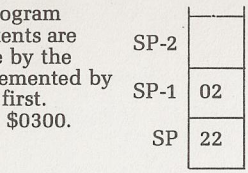
Subroutines

Program control is transferred to a subroutine by using the instruction JSR. This requires 3 bytes of machine code using the absolute addressing mode.

Example: 2200 JSR $0300
2203 ----------

When the instruction is executed the program counter is incremented by 2 and its contents are pushed onto the stack one byte at a time by the CPU. Note that the stack pointer is decremented by 2 and that the high-order byte is stored first.

Program control is then transferred to $0300.



Subroutines are normally terminated by the instruction RTS (return from subroutine). This is a 1-byte instruction and when executed the stack pointer is incremented by 1 and the byte stored in the stack is loaded into the least significant byte of the program counter; the stack pointer is incremented again and the byte stored in the stack is loaded into the most significant byte of the program counter. Then the program counter is incremented by 1 and program control passes to this address.

Thus in the previous example program control would be transferred to $2203, the next instruction in the program sequence to be executed.

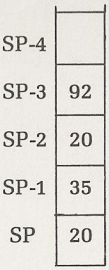
It should be noted that on entering a subroutine the status of the CPU is not saved. Therefore it may be necessary to save the contents of the accumulator, the processor status register and the index registers on the stack.

Nested subroutines

It is permissible to use nested subroutines.

```
Example   2033                JSR SUB1      ; main program
          2036                -------
          2080      SUB1      -------       ; subroutine 1
                    -------
          2090                JSR SUB2
                    ------
                                RTS
          3000      SUB2      -------       ; subroutine 2
                                RTS
```

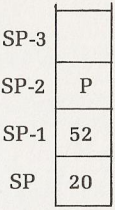
Immediately prior to the execution of the RTS instruction in subroutine 2 the stack would contain:



Interrupts

When an interrupt service routine is entered the CPU pushes the contents of the return address of the main-line program (high-order byte first) and the contents of the processor status register (P) onto the stack in that order.

Example: 2050 LDA #MEM ← interrupt occurs here
2052



Note that at the time of the interrupt the current instruction is completed and the program counter will contain the address of the next instruction.

Interrupt service routines are normally terminated with the RTI instruction (return from interrupt). This is a 1-byte instruction and when executed will increment the stack pointer and pull the contents of the processor status register and the program counter contents from the stack. In the example given, program control is transferred back to location $2052.

Stack manipulation

Data stored in the stack may be changed by the programmer by incrementing or decrementing the stack pointer to the required location and using the push and pull instructions. There are no instructions which directly increment or decrement the stack pointer but use is made of the index register X.

TSX
DEX or INX
TXS

This routine transfers the contents of the stack pointer to the index register which is then incremented or decremented as required and is transferred back to the stack pointer.

The stack pointer can also be changed by using the push and pull instructions; care must be taken not to corrupt the stack or the accumulator.

Input/Output

There are a number of input/output devices which are compatible with the 6502 microprocessor. It is beyond the scope of this Guide either to give details or to list all of these devices. A list of the more common ones is shown below and full details are given separately on the aptly named Versatile Interface Adapter (VIA).

(a) PERIPHERAL INTERFACE ADAPTER (PIA) 6520
(b) ASYNCHRONOUS COMMUNICATIONS INTERFACE ADAPTER (ACIA) 6551
(c) CRT CONTROLLER (CRTC) 6545
(d) ROM-RAM-I/O-COUNTER (RRIOC) 6531
(e) ROM-I/O-COUNTER (RIOC) 6534
(f) ROM-RAM-I/O-INTERVAL TIMER (RRIOT) 6530
(g) RAM, I/O,INTERVAL TIMER (RIOT) 6532
(h) VERSATILE INTERFACE ADAPTER (VIA) 6522

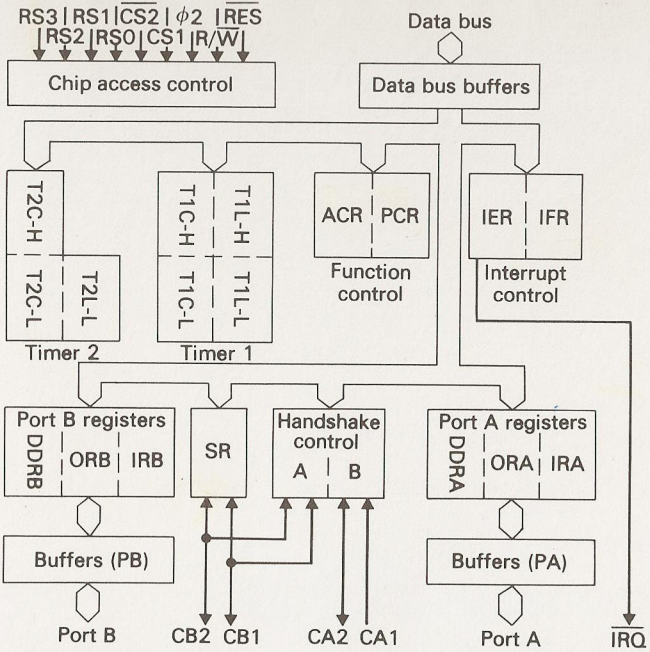
The manufacturers' data sheets should be consulted for the full specification, pin connections, electrical characteristics and timing diagrams.

Versatile Interface Adapter (VIA) 6522

The Versatile Interface Adapter contains the following features:

(a) two eight-bit bidirectional input/output ports
(b) two interval timers
(c) serial to parallel and parallel to serial shift register
(d) four peripheral control lines, CA1, CA2, CB1 and CB2
(e) interrupt facilities.

Figure 2 shows the block diagram of the 6522 VIA internal registers and the relevant pin connections.



Key			
T2C-H	Counter	ACR	Auxiliary
T2C-L	Counter	PCR	Peripheral
T2L-L	Latch	IER	Enable
T1C-L	Counter	IFR	Flags
T1C-L	Counter	SR	Shift register
T1L-H	Latch	A	Port A
T1L-L	Latch	B	Port B
DDRB	Data dir.	DDRA	Data dir.
ORB	Output	ORA	Output
IRB	Input latch	IRA	Input latch

Fig. 2 R6522 Block Diagram

Internal registers

The VIA has 16 8-bit memory-mapped programmable internal registers. The registers are selected by the chip select lines CS1 (active high) and CS2 (active low) with the 4-register select lines, RS0, RS1, RS2 and RS3. The table below shows the decoding of the 16 registers.

Register No. $	RS3	RS2	RS1	RS0	Register Mnemonic	Description	
						Write	Read
0	0	0	0	0	ORB/IRB	Output register B	Input register B
1	0	0	0	1	ORA/IRA	Output register A	Input register A
2	0	0	1	0	DDRB	Data direction register B	
3	0	0	1	1	DDRA	Data direction register A	
4	0	1	0	0	T1C-L	T1 low-order latch	T1 low-order counter
5	0	1	0	1	T1C-H	T1 high-order counter	
6	0	1	1	0	T1L-L	T1 low-order latch	
7	0	1	1	1	T1L-H	T1 high-order latch	
8	1	0	0	0	T2C-L	T2 low-order latch	T2 low-order counter
9	1	0	0	1	T2C-H	T2 high order counter	
A	1	0	1	0	SR	Shift register	
B	1	0	1	1	ACR	Auxiliary control register	
C	1	1	0	0	PCR	Peripheral control register	
D	1	1	0	1	IFR	Interrupt flag register	
E	1	1	1	0	IER	Interrupt enable register	
F	1	1	1	1	ORA/IRA	No handshake mode	

Note: When the VIA is reset, the data direction registers DDRA and DDRB the control registers PCR and ACR and the interrupt flag register IFR are all cleared. Interrupts are also disabled.

Input/output ports

Each port has 8 peripheral data lines (PA7-PA0, PB7-PB0) which can be configured individually as either input or output using the Data Direction Registers DDRA and DDRB.

Data direction registers DDRB and DDRA (Registers $2 and $3)

A peripheral data line is established as an input by writing a logic 0 to the corresponding bit of the data direction register; it is established as an output by writing a logic 1.

Output registers ORB and ORA (Registers $0 and $1)

If a peripheral data line is established as an output, a logic 1 written to the corresponding bit in the output register will cause the data line to go high; a logic 0 causes the line to go low.

When reading a peripheral data line on port A that has been established as an output, the voltage level on the line determines whether it is read as a logic 0 or 1; however for a line on port B, the bit stored in the output register ORB determines the value read.

Input Registers IRB and IRA (Registers $0 and $1)

If a line is established as an input, a CPU read of the corresponding bit in the input register will be logic 1 if the voltage level on the line corresponds to a standard TTL logic 1; if the voltage level corresponds to a TTL logic 0 then a logic 0 is read.

If an input line is left unconnected it will be read as a logic 1.

Example: Configure port A as an input port and port B as an output port; read the input data on port A and display on port B.

```
LDA #$00    ; load 0000 0000 into the data direction register side
              A
STA DDRA    ; all lines on side A configured as input
LDA #$0FF   ; load 1111 1111 into the data direction register side
              B
STA DDRB    ; all lines on side B configured as output
LDA IRA     ; read data on port A
STA ORB     ; display data on port B
```

Further example: Configure PA7 and PA6 on side A as input and lines PA5 through PA0 as output.

```
LDA #$3F; load 0011 1111 into the data direction register side A
STA DDRA; PA7-PA6 input, PA5-PA0 output
```

Peripheral control register (Register $C)

The peripheral control register PCR selects the operating modes for the control lines, CA1, CA2, CB1 and CB2; its format is shown in Fig. 3.

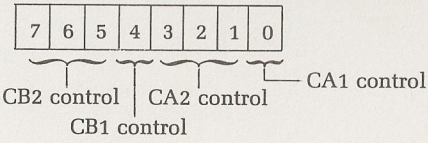


Fig. 3

Control line CA1

CA1 can be configured only as an input line; bit 0 of the PCR defines the active transition of CA1 as follows:

0 high to low transition
1 low to high transition

An active transition on CA1 sets bit 1 of the interrupt flag register (IFR). The flag can be reset by an CPU read or write of the output register ORA.

Control line CA2

CA2 is a bidirectional line and is configured by bits 1, 2 and 3 in the PCR.

CA2 established as an input: bit 3=0
bit 1 defines the type of input mode:
0 normal input mode
1 independent interrupt mode

bit 2 defines the active transition of CA2:
0 high to low transition
1 low to high transition

An active transition of CA2 sets bit 0 of the interrupt flag register (IFR).

For normal input mode, the interrupt flag CA2 of the interrupt flag register (IFR) is reset by a CPU read or write of the output register ORA. For the independent interrupt mode the CA2 flag can only be reset by a write to the interrupt flag register (*see* Interrupt facilities p. 58).

CA2 established as an output: bit 3=1

bit 2	bit 1	
0	0	CA2 goes low on an CPU read or write of the output register ORA; it is returned high by the next active transition of CA1. This is called the handshake mode.
0	1	CA2 goes low on an CPU read or write of the output register ORA; it is returned high on the next negative transition of the system clock pulse. This is referred to as the pulse mode.
1	0	CA2 is reset.
1	1	CA2 is set.

The control lines CB1 and CB2

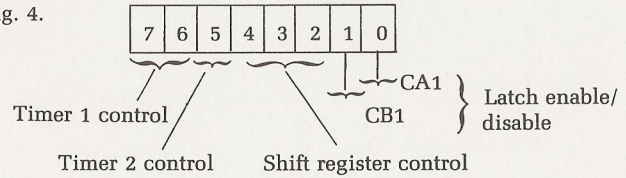
The control lines CB1 and CB2 are configured almost identically to CA1 and CA2; bit 4 configures CB1 and bits 5, 6 and 7 configure CB2 in the same way that bits 0 and 1, 2 and 3 configure CA1 and CA2.

There are however small differences: (a) CB1 can be used as an output line (unlike CA1) for use in the shift register mode; (b) when CB2 is established as an output in the handshake or pulse mode, it will go low only on a CPU write to the output register ORA.

Auxiliary control register (register $B)

The auxiliary control register configures the two timer counters, 1 and 2 and the shift register; it also provides the facility for enabling and disabling the latching on ports A and B. its format is shown in Fig. 4.

Fig. 4.



Latching of data

The data on ports A and B can be latched at the time of the last active transition on control lines CA1 and CB1. In order to configure the latching mode, bits 0 and 1 of the auxiliary control

register must be logic 1 and to disable they must be logic 0. As soon as there is a CPU read of the appropriate input register ORA or ORB the latch is disabled.

Example: Configure the VIA to enable data to be latched on port A but to be disabled on port B.

```
LDA #$01      ; bit pattern 0000 0001
STA ACR       ; port A latching enabled, port B disabled.
```

When there is an active transition of CA1 (defined by bit 0 of the peripheral control register) the data appearing at the port A at the time of the active transition will be latched. The next read of the register ORA will be the latched data and not the data appearing currently on the port.

Timer 1

Timer 1 has a 16-bit counter and 8-bit low-order and high-order latches. There are 4 modes of operation and are configured by bits 6 and 7 of the Auxiliary Control Register ACR.

bits		
7	6	
0	0	Timed interrupt mode single shot with PB7 disabled: This mode generates a time interval equal to N clock pulses where N is the number loaded into the T1 counter. Following the CPU write to T1C-H the interrupt flag T1 of the interrupt flag register IFR is reset and the counter is decremented down to zero at the system clock rate. At this point the interrupt flag is set and the $\overline{IRQ}$ line will go low if $\overline{IRQ}$ interrupts are enabled. The counter will continue to decrement beyond zero if interrupts have been enabled. The time interval since the interrupt can be determined by a CPU read of the counter.
0	1	Continuous interrupts with PB7 disabled: This mode generates continuous interrupts whose periods equal N system clock pulses where N is the number loaded into the T1 counter. Following the CPU write to T1C-H the interrupt flag T1 of the interrupt flag register IFR is reset and the counter is decremented down to zero at the system clock rate. When the count reaches zero the interrupt flag is set, the contents of the latches are automatically transferred to the counter and the process is automatically repeated.

1 0 Single shot output on PB7:
This mode generates a single negative pulse on the peripheral data pin PB7 of duration equal to N clock pulses where N is the number in the T1 counter. Following a CPU write to T1C-H, PB7 goes low and is returned high when the counter is decremented to zero. PB7 must be configured as an output line in the data direction register DDRB.

1 1 Continuous output on PB7:
This mode generates a continuous wave form on PB7. Following a CPU write to T1C-H, PB7 goes low and is returned high when the counter is decremented to zero the interrupt flag being set. Then the contents of the latches are automatically transferred to the counter, the interrupt flag is reset and the counter is decremented to zero when PB7 is inverted; the process is repeated continuously.

Note: For precise timing wave forms, see manufacturer's data sheet.

CPU read and write of T1 counter

The 16-bit counter is loaded by the following sequence: the low-order byte is loaded into the T1 low-order latch by a CPU write to T1C-L; the high-order byte is loaded into the T1 high-order latch by a CPU write to T1C-H.

Following the CPU write to T1C-H, the contents of both the low and high-order latches are transferred into the T1 counter; the interrupt flag T1 is reset and the counter is automatically decremented at the system clock rate.

Example: load the timer counter T1 with $8024.

```
LDA #$24    ;
STA T1C-L   ; load the low-order latch first with $24
LDA #$80    ; load the high-order latch with $80, the counter is
              then loaded and the T1 interrupt flag is reset.
STA T1C-H   ; The counter now decrements to zero at the system
              clock rate.
```

The 16-bit counter is read by the following sequence:
The low-order counter is read by a CPU read of T1C-L, T1 interrupt flag is rset, the high-order counter is read by a CPU read of T1C-H.

Example: read the contents of T1 and store in memory locations called MEM and MEM+1.

```
LDA T1C-L    ; low-order byte is transferred to MEM;
STA MEM      ; T1 flag is reset
LDA T1C-H    ; high-order byte is transferred to MEM+1
STA MEM+1
```

CPU read and write of latches

The low-order and high-order latches may read and be written to by;

```
LDA T1L-L    ; CPU read of low-order latch
STA T1L-L    ; CPU write to low-order latch
LDA T1L-H    ; CPU read of high-order latch
STA T1L-H    ; CPU write to high-order latch
```

No transfer to the counter takes place and the interrupt flag T1 remains unaffected by these read/writes.

Timer 2

Timer 2 consists of a 16-bit counter T2 and a low-order latch; there are two modes of operation configured by bit 5 in the auxiliary control regi ter ACR.

Bit 5

0 Timed interrupt mode:
This mode generates a time interval equal to N system clock pulses where N is the number loaded into the counter T2. Following a CPU write to T2C-H the counter is decremented to zerom

1 Pulse counting mode:
In this mode the peripheral data pin PB6 is used to count a predetermined number of pulses. The number of pulses to be counted is loaded into the counter T2; the CPU write to T2C-H clears the interrupt flag T2 and the counter is decremented every time a pulse appears on PB6. When the count is zero the interrupt flag is set.
Note: PB6 must be configured as an input.

CPU read and write of T2 counter

The 16-bit counter is loaded by the following sequence:
The low-order byte is loaded into the T2 low-order latch by an CPU write to T2C-L; the high-order byte is loaded into the T2 high-order counter by a CPU write to T2C-H.

Following the CPU write to T2C-H the contents of the low-order latch are automatically transferred to the low-order counter, the interrupt flag T2 is reset and the counter is decremented.

The counter is read by the following sequence:
The low-order byte is read by a CPU read of T2C-L, the T2 interrupt flag is reset and the high-order byte is read by a CPU read of T2C-H.

Shift register

The shift register mode is configured by bits 4, 3 and 2 of the auxiliary control register ($B). There are seven modes of operation together with a disabling mode. Data is shifted serially in or out of the shift register (SR) via the peripheral control line CB2 under the control of internal or external shift pulses on the line CB1. The rate of data transfer depends on the mode selected.

The modes of operation are summarized as follows:

bit 4 3 2	CB1	CB2	
0 0 0	–	–	the shift register is disabled and the control lines CB1 and CB2 are controlled by peripheral control register PCR; the interrupt flag is reset.
0 0 1	o/p	i/p	the rate of data transfer is determined by the T2 counter and the system clock period. Data transfer is initialized by a CPU read or write of the shift register; the interrupt flag is set after eight CB1 shift pulses.
0 1 0	o/p	i/p	the rate of data transfer is determined by the system clock period. Data transfer is initialized by a CPU read or write of the shift register; the interrupt flag is set after eight CB1 shift pulses.
0 1 1	i/p	i/p	the rate of data transfer is determined by an external device. The interrupt flag is set after eight CB1 shift pulses.
1 0 0	o/p	o/p	the rate of data transfer is determined by the T2 counter and the system clock period. Data transfer is initialized by a CPU read or write of the shift register.
1 0 1	o/p	o/p	the rate of data transfer is determined by the T2 counter and the system clock period.

			Data transfer is initialized by a CPU read or write of the shift register. The interrupt flag is set after eight CB1 shift pulses.
1 1 0	o/p	o/p	the rate of data transfer is determined by the system clock period. Data transfer is initialized by a CPU read or write of the shift register. The interrupt flag is set after eight shift pulses.
1 1 1	i/p	o/p	the rate of data transfer is determined by an external device. Data transfer is initialized by a CPU read or write of the shift register. The interrupt flag is set after eight shift pulses.

Interrupt facilities

The 6522 VIA has a single interrupt request line $\overline{\text{IRQ}}$ which is active low. This line is enabled or disabled by the state of both the interrupt flag register IFR and the interrupt enable register IER.

Interrupt flag register (register $D)

The VIA has an interrupt flag register containing 6 interrupt flags and a bit IRQ which indicates that an $\overline{\text{IRQ}}$ interrupt has been enabled; the format for the flag register is shown in Fig. 5.

Fig. 5.

IRQ	T1	T2	CB1	CB2	SR	CA1	CA2

IRQ this bit is set following any enabled interrupt (it reflects the status of the $\overline{\text{IRQ}}$ (active low) interrupt line of the VIA).

T1 this flag is set when the counter T1 has been decremented down to zero; it is reset by a CPU read of T1C-L or a write to T1C-H.

T2 this flag is set when the counter T2 has been decremented down to zero; it is reset by a CPU read of T2C-L or a write to T2C-H

CB1 this flag is set following an active transition of CB1; it is reset by a CPU read or write of ORB

CB2 this flag is set following an active transition on CB2; it is reset by a CPU read or write of ORB

SR this flag is set on the completion of 8 shifts; it is reset by a CPU read or write of the shift register.

CA1 this flag is set following an active transition on CA1; it is reset by a CPU read or write of ORA

CA2 this flag is set following an active transition on CA2; it is reset by a CPU read or write of ORA

Note: Individual flags can also be reset by writing a logic 1 in the appropriate bit in the IFR.

Interrupt enable register

$\overline{IRQ}$ interrupts can be disabled or enabled by an CPU write to the interrupt enable register whose format is similar to that of the flag register with the exception of bit 7 (set/clear). (See Fig. 6.)

Fig. 6

7	6	5	4	3	2	1	0
S/C	T1	T2	CB1	CB2	SR	CA1	CA2

Interrupts may be disabled by writing a logic 0 to bit 7 together with a logic 1 to the appropriate bit.

Example: LDA #$03 ; 0000 0011
STA IER ; disables CA1 and CA2 interrupts.

Interrupts may be enabled by writing a logic 1 to bit 7 with a logic 1 to the appropriate bit

Example: LDA #$92 ; 1001 0010
STA IER ; enables CB1 and CA1 interrupts

If a CPU read of the IER is made, bit 7 will always appear as a logic 1 and the other bits will reflect their enabled (logic 1) or disabled (logic 0) state.

Interrupt operation

An $\overline{IRQ}$ interrupt will occur if any flag in the IFR is set *and* the corresponding bit in the IER has been previously enabled. When bit 7 in the interrupt flag register is high then this is an indication that the $\overline{IRQ}$ line has gone low.

Interrupts and reset

The 6502 microprocessor has two hardware interrupt lines: NMI (non-maskable interrupt) and $\overline{IRQ}$ (interrupt request). Both lines are active low but $\overline{IRQ}$ is level sensitive whilst NMI is edge sensitive.

The addresses for the two interrupt service routines are stored in the following memory locations

FFFA	low-order byte	$\overline{\text{NMI}}$
FFFB	high-order byte	
FFFC	low-order byte	$\overline{\text{RES}}$
FFFD	high-order byte	
FFFE	low-order byte	$\overline{\text{IRQ}}$ and BRK
FFFF	high-order byte	

The address for the reset routine is included and is discussed in the section on Reset; there is a software interrupt instruction BRK which has the same interrupt service routine address as $\overline{\text{IRQ}}$.

Interrupt request (IRQ)

This line is normally connected to the interrupt request lines of input/output devices; in the case of the VIA this is $\overline{\text{IRQ}}$. The interrupt can be disabled by setting the interrupt disable bit in the processor status register to logic 1 and enabled with a logic 0 (see SEI and CLI instructions). When the $\overline{\text{IRQ}}$ line goes low an interrupt is requested and the following sequence takes place:

1 The current instruction in the main line program is completed.
2 The I bit in the processor status register is polled and if it is equal to 1 the interrupt is ignored and the next instruction in the main line sequence will be executed; an interrupt flag will have been set as a result of the request and this can later be interrogated or cleared as appropriate.
3 If the I bit is not set then the CPU pushes the contents of the program counter on to the stack (high-order byte first) together with the contents of the processor status register.
4 The interrupt disable bit is set to 1 to prevent further $\overline{\text{IRQ}}$ interrupts and break command flag is set.
5 The address of the interrupt service routine is obtained from the addresses $FFFE and $FFFF and loaded into the program counter thus transferring program control to the interrupt service routine.

Non-maskable interrupt ($\overline{\text{NMI}}$)

A negative transition on this line generates an interrupt that cannot be masked or disabled; the following sequence takes place:

1 The current instruction in the main line program is completed.
2 The CPU pushes the contents of the program counter onto the stack (high-order byte first) together with the contents of the processor status register.

3 The interrupt disable bit is set to 1 to prevent $\overline{IRQ}$ type interrupts and the break command flag is set to 1.
4 The address of the interrupt service routine is obtained from the addresses $FFFA and $FFFB and loaded into the program counter thus transferring program control to the interrupt service routine.

Note that the $\overline{NMI}$ interrupt has a higher priority than $\overline{IRQ}$.

Saving CPU status

The contents of the accumulator and the index registers are not saved on entering an interrupt service routine. This can be accomplished by:

```
PHA ; push accumulator contents onto the stack
TYA ;
PHA ; push index register Y onto the stack
TXA ;
PHA ; push index register X onto the stack
```

The data can be retrieved before leaving the interrupt service routine by:

```
PLA ; pull data from stack into accumulator
TAX ; transfer contents of accumulator into the index register X
PLA ; pull data from stack into accumulator
TAY ; transfer contents of accumulator into the index register Y
PLA ; pull contents of accumulator from the stack
```

Return from interrupt

The interrupt service routines for both $\overline{IRQ}$ and $\overline{NMI}$ are normally terminated by the instruction RTI. This instruction increments the stack pointer by 1 and retrieves the previously stored processor status register; the stack pointer is incremented again to retrieve the contents of the program counter and thereby transfers program control back to the main-line program.

Multiple $\overline{IRQ}$ interrupts

The majority of input/output devices have more than one source of $\overline{IRQ}$ interrupts; the 6522 VIA has 7 interrupt modes and if any one is enabled program control will be transferred to the same $\overline{IRQ}$ interrupt service routine. The source of the interrupt can be determined by polling the interrupt flag register immediately on entering the interrupt service routine. The order in which the flags

are polled depends on the priority decided by the programmer. If more than one peripheral input/output device is interfaced to the CPU then the principle is still the same but more time will be used to poll the flag registers of the different devices.

Reset

This line is used to reset or initialize the microprocessor from a power down situation. When the line is taken low all CPU read/writes cease, and when the line is taken high the following sequence takes place:

1 The interrupt disable bit in the processor status register is set.
2 The address of the reset routine is obtained from the memory locations $FFFC and $FFFD and loaded into the program counter thus transferring program control to the reset routine.

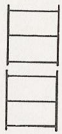
This line is often used to retrieve the situation in the event of system runaway; depression of the reset key will normally bring the system back to its initialization condition.

BRK

The break command instruction is a software interrupt, and upon execution the break command flag in the processor status register is set. Then the CPU pushes the contents of the program counter (high-order byte first) and the processor status register onto the stack. The interrupt disable bit is set to 1 to prevent $\overline{IRQ}$ interrupts. The address of the interrupt service routine is obtained from the addresses $FFFE and $FFFF and is loaded into the program counter thus transferring control to the interrupt service routine. In order to distinguish between an $\overline{IRQ}$ and BRK interrupt, the processor status register must be pulled from the stack and the break command flag interrogated.

Example: 8-bit binary addition. The two numbers to be added are stored in locations MEM and MEM+1 and the answer in locations ANSWRL and ANSWRH.

```
   CLD            ; binary arithmetic
   CLC            ; clear carry
   LDA #$00       ;
   STA ANSWRH     ; clear high-order byte      MEM
   ADC MEM        ; first number               MEM+1
   ADC MEM+1      ; second number
   STA ANSWRL     ; store low-order byte       ANSWRL
   BCC CONT       ; carry clear                ANSWRH
   INC ANSWRH     ; increment high-order byte
CONT ------       ; continue
```



Note: (a) The 6502 instruction set does not have an 'add without carry' instruction. The carry flag bit must be cleared before the start of an addition.
(b) The decimal mode flag must be cleared so as to ensure that binary arithmetic is performed.

Example: 8 bit binary subtraction. The minuend is stored in location MEM and the subtrahend in MEM+1; the answer is in locations ANSWRL and ANSWRH.

```
   CLD              ; binary arithmetic
   SEC              ; set carry
   LDA #$00         ;
   STA ANSWRH       ; clear high-order byte
   LDA MEM          ; minuend
   SBC MEM+1        ; subtrahend
   STA ANSWRL       ; store low-order byte
   BCS CONT         ; carry set then branch
   DEC ANSWRH       ; decrement high-order byte
CONT ------         ; continue
```

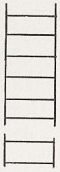
Note: (a) The carry flag has to be set prior to a subtraction. When the subtrahend is greater than the minuend, and as a consequence there is a 'borrow', then the carry flag is reset and *not* set.
(b) The decimal mode flag must be cleared to ensure that binary subtraction is performed.

Example: 16-bit binary addition. The two 16-bit binary numbers to be added are stored in locations NUMB1H, NUMB1L, NUMB2H, NUMB2L; the answers are deposited in locations ANSWRL, ANSWRM and ANSWRH.

```
   CLD              ; binary arithmetic
   CLC              ; clear carry
   LDA #$00         ;
   STA ANSWRH       ; clear high-order byte
   ADC NUMB1L       ;
   ADC NUMB2L       ; add low-order bytes
   STA ANSWRL       ; store low-order byte
   LDA #$00         ; clear accumulator
   ADC NUMB1H       ;
   ADC NUMB2H       ; add high-order byte
   STA ANSWRM       ; store middle-order byte
   BCC CONT         ; carry clear then branch
   INC ANSWRH       ; increment high-order byte
CONT ------         ; continue
```

Example: 8-bit multiplication using the 'shift and add' technique. The multiplier is stored in location MULP and the multiplicand in MULT. The answer is stored in ANSWRL and ANSWRH. Count holds the number of shift and adds.

```
   CLD              ; binary arithmetic
   LDA #$00         ; clear memory locations
   STA MULTH        ;
   STA ANSWRL       ;
   STA ANSWRH       ;
   LDA #$08         ;
   STA COUNT        ; set count to 8
AGAIN ROR MULP      ; examine first bit
   BCC SHIFT        ; branch for no addition
   CLC              ; clear carry
   LDA MULTL        ;
   ADC ANSWRL       ; add low-order
   STA ANSWRL       ; store low-order byte
   LDA MULTH        ;
   ADC ANSWRH       ; add high-order
   STA ANSWRH       ; store high-order byte
SHIFT ASL MULTL     ; shift MULT for next addition
   ROL MULTH        ;
   DEC COUNT        ; count=count-1
   BNE AGAIN        ; next bit
CONT ------         ; continue
```



Example: on $\overline{\text{IRQ}}$ interrupt: configure port B of a VIA as output and port A as input, enable CA1 interrupt and disable all others.

```
   SEI              ; set the interrupt mask
   LDX #$FF
   TXS              ; load the stack pointer
   LDA #$00         ; clear A
   STA PCR          ; active transition on CA1 1→0
   STA DDRA         ; port A configured as input
   LDA #$FF
   STA DDRB         ; port B configured as output
   LDA #$7F
   STA IER          ; disable all interrupts
   LDA #$82         ;
   STA IER          ; enable CA1 interrupts
   CLI              ; clear interrupt mask
   ------           ; continue with main line program
```

Interrupt service routine

```
   LDA IRA ; clear CA1 flag in the VIA flag register
------- ; continue with interrupt service routine
-------
RTI     ; terminate interrupt service routine
```

Note the following:
(a) The stack pointer is initialized; failure to do this may cause program corruption.
(b) The interrupt disabled bit is set during the initialization procedure to prevent $\overline{IRQ}$ interrupts; it is cleared prior to the continuation of the main-line program.
(c) The first instruction in the interrupt service routine could be replaced by STA ORA in order to clear the CA1 flag.
(d) The memory locations $FFFE and $FFFF must hold either directly or indirectly the address of the interrupt service routine.
(e) If there is a likelihood of a BRK instruction being executed to cause an interrupt, the processor status register should be retrieved from the stack at the start of the interrupt service routine and the break command flag interrogated.

Example: Configure Timer 1 for continuous square-wave output on PB7.

```
   LDA #$7F      ; 0111 1111
   STA IER       ; disables all interrupts
   LDA #$FF      ; 1111 1111
   STA DDRB      ; side B configured as output
   LDA #$C0      ; 1100 0000
   STA ACR       ; mode 11 selected, continuous o/p on PB7
   LDA #$mm      ;
   STA T1C-L     ; load low-order latch
   LDA #$nn      ; load counter T1 interrupt flag reset
   STA T1C-H     ; counter decremented to zero
CONT ------      ; continue
```

Note: (a) Following CPU write to T1C-H, the peripheral data line PB7 goes low and is returned high when the counter is decremented to zero, i.e. after $nnmm system clock pulses; the latches are then loaded into the counter and the process is repeated.
(b) By modifying the contents of the ACR to #$80 instead of #$C0, mode 01 is selected and a single negative pulse is produced on PB7.

Example: Configure Timer 2 to count a predetermined number of pulses on PB6; disable T2 interrupts.

```
   LDA #$7F        ; 0111 1111
   STA IER         ; disables all interrupts
   LDA #$BF        ; 1011 1111
   STA DDRB        ; bit 6 i/p, other bits o/p
   LDA #$20        ; 0010 0000
   STA ACR         ; bit 5 = 1 in ACR, mode 1 selected
   LDA #$mm        ; low-order latch loaded
   STA T2C-L       ;
   LDA #$nn        ; high-order counter loaded, low order latch
   STA T2C-H       ; transferred to low-order counter, interrupt
                     flag is reset
LOOP LDA IFR       ; read interrupt flag register
   AND #$20        ; isolate bit 5
   BEQ LOOP        ; repeat if not set
```

Note: (a) The peripheral data line PB6 must be configured as an output.
(b) When $nnmm pulses have been counted on PB6, the interrupt flag is set.
(c) $\overline{IRQ}$ interrupts are disabled.

Example: Generate on CB2 a pulse waveform whose frequency is 500 Hz and has a mark-space ratio of 1:1.

```
AGAIN LDA #$EO; 1110 000
   STA PCR     ; CB2 set to logic 1
   JSR DELAY   ;
   LDA #$C0    ; 1100 0000
   STA PCR     ; CB2 set to logic 0
   JSR DELAY   ;
   JMP AGAIN   ; repeat waveform
DELAY ...      ; delay routine 1ms
      ...
      RTS
```

Example: Analogue to digital conversion

A 10-bit A/D is interfaced to a VIA; it requires a convert pulse (a 1→0→1 transition) and its status goes low on the completion of the conversion. CA1 is used to monitor the status and CA2 generates the start pulse. The 10-bit digital word is stored in memory locations MEM and MEM+1. *See* Fig. 7

```
   LDA #$00        ; clear A
   STA DDRA        ; configure port A input
   STA DDRB        ; configure port B input
```

```
  STA PCR          ; configure 1 → 0 active transition on CA1
  LDA #$0E
  STA PCR          ; 0000 1110 set CA2
  LDA #$0C
  STA PCR          ; 0000 1100 reset CA2
  LDA #$0E
  STA PCR          ; 0000 1110 set CA2
WAIT LDA IFR       ; read flag register
  AND #$02         ; mask CA1 flag
  BNE WAIT         ; conversion finished ?
  LDA IRA          ; yes read 8 least significant bits
  STA MEM+1        ; store
  LDA IRB          ; read port B
  AND #$03         ; read bits 9 and 8
  STA MEM          ; store
```

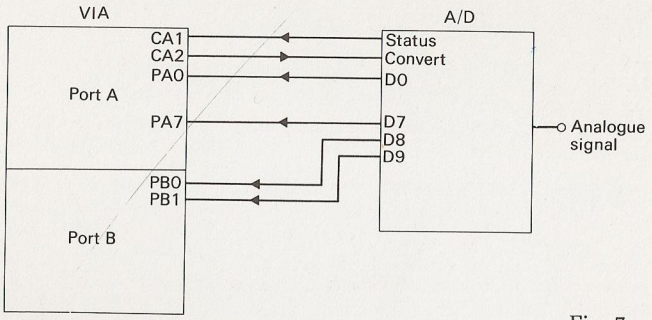


Fig. 7.

Example: Configure a number of VIAs so that all the port As are output using indexed indirect addressing. The table in page $00 of memory contains the addresses of the data direction registers.

```
  LDX #$00    ;
  LDA #$0FF   ; bit pattern 1111 1111 for output
  STA ($80,X) ; DDRA1 configured
  STA ($82,X) ; DDRA2 configured
STA ($84,X)   ; DDRA 3 configured
  STA ($86,X) ; DDRA4 configured
```

0080	DDRA1l
0081	DDRA1h
0082	DDRA2l
0083	DDRA2h
0084	DDRA3l
0085	DDRA3h
0086	DDRA4l
0087	DDRA4h

Example: Output through port B of a VIA the contents of a table in sequence, using indirect indexed addressing.

```
        LDY #$00
NEXT    LDA ($40),Y     ; start of table
        STA ORA         ; output state
        JSR DELAY       ; delay
        INY             ; increment table pointer
        CPY $42         ; end of table
        BNE NEXT        ; no next state

DELAY   -------         ; delay routine
        -------
        RTS
```

0040	00
0041	30
0042	08
3000	0C
3001	04
3002	06
3003	02
3004	03
3005	01
3006	09
3007	08

Note: (a) the address of the table is stored in page $00 locations $40 and $41.
(b) the number of elements in the table is stored in $42
(c) the data is stored in the table $3000–$3007

By changing the page $00 data, another table of values can be accessed without altering the main program.

Example: 8 switches are connected to port A of a VIA; by using the various logic instructions determine the state of individual switches

SW7	SW6	SW5	SW4	SW3	SW2	SW1	SW0

Method 1: use the logical AND instruction to test SW1

```
LDA IRA         ; read all the switches
AND #$02        ; accumulator contains 0000 00X0, use BNE or
                  BEQ to test Z flag
```

Method 2: use the rotate ROR instruction to test SW0

```
LDA IRA         ; read all the switches
ROR A           ; rotate right once into the carry flag, use BCC or
                  BCS to test C flag
```

Method 3: use the instructions BPL and BMI to test SW7

```
LDA IRA         ; N flag reflects the state of bit 7, test N with BPL or
                  BMI
```

Method 4: use of the BIT function to test switch 7, 6 and n

```
LDA #$08        ; bit pattern 0000 1000
```

```
BIT IRA          ; logical AND performed between IRA and
                   accumulator, Z flag can be used to test bit n (in
                   this case 3). N and V flags loaded with bits 7 and
                   6; use BVS and BVC.
```

A combination of these techniques can be used, depending on the particular application.

Other programming books available from Pitman

BASIC: A Short Self-instructional Course
M J Oatey and C Payne

FORTRAN Reference Manual
P F Ridler

Introduction to BASIC
J B Morton

Methodical Programming in COBOL
R Welland

Pascal (second edition)
W Findlay and D A Watt

Pascal for Science and Engineering
J McGregor and A Watt

Principles of Programming: An Introduction with FORTRAN
E B James

Simple Pascal
J McGregor and A Watt

Structured BASIC and Beyond
W Amsbury

Structured Programming: A Self-instruction Course
R Thurner